Tries that match

Sebastian Graf
Karlsruhe Institute of Technology
Karlsruhe, Germany
sebastian.graf@kit.edu

Abstract

In applications such as compilers and theorem provers, we often want to match a target term against multiple patterns (representing rewrite rules or axioms) simultaneously. Efficient matching of this kind is well studied in the theorem prover community, but much less so in the context of statically typed functional programming. Doing so is the topic of this talk and yields an interesting new viewpoint – one marrying up datatype-generic derivation of generalized tries [Hinze 2000] with datatype-generic operations for using said tries as a term index data structure, providing efficient matching lookup of a query term against indexed patterns.

We start by reviewing Ralph Hinze's work „Generalizing generalized tries" [Hinze 2000] as an introduction to both polytypic (or datatype-generic) programming in Haskell and *generalized tries*: Tries keyed by data types instead of strings of symbols. After studying examples for some concrete data types and their corresponding lookup function, we examine how Hinze manages to derive generalized tries for arbitrary first-order polymorphic datatypes.

After this review, we discuss how to extend the polytypic formulation to cope with syntax trees, that is, datatypes with variable occurrences and binding constructs. The challenge is in performing lookup modulo alpha-equivalence by doing de Bruijn numbering on the fly, as well as in encoding the binding constructs in the datatype-generic sums-of-products schema language.

Finally we go beyond one-to-one, exact lookups by storing (pattern,value) associations in our trie data structure, where each pattern may specify and mention a number of (first-order) unification variables. We use this trie as a *term index*, where a query with a given ground term against the index retrieves all (pattern,value) pairs of which the query term is a unification instance. Our unique achievement is that the query term is not copied during look up.

As expected, performance measurements indicate that our tries can significantly outperform ordered maps and hash maps for exact lookups of terms. We haven't yet put our matching triemap implementation to use in the Glasgow Haskell Compiler (GHC), but eventually plan to do so, replacing linear searches in the process.

We conclude by comparing our encoding of a term index to other Haskell libraries, as well as to *discrimination trees* [McCune 1992], the encoding that is closest to our trie data structure in automated reasoning literature. We'll see that the dataype-generic encoding in Haskell admits (minor) safety guarantees compared to the untyped string representation of the term the discrimination tree operates on. The generic encoding also allows for bespoke data structures depending on the term sort.

ACM Reference Format:
Sebastian Graf. 2022. Tries that match. *Proc. ACM Program. Lang.* 1, POPL, Article 1 (January 2022), 1 page.

lkjlkj

References

Ralf Hinze. 2000. Generalizing Generalized Tries. *Journal of Functional Programming* 10, 4 (2000), 327–351. http://journals.cambridge.org/action/displayAbstract?aid=59745

William McCune. 1992. Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. *J. Autom. Reason.* 9, 2 (oct 1992), 147–167. https://doi.org/10.1007/BF00245458

Author's address: Sebastian Graf, Karlsruhe Institute of Technology, Karlsruhe, Germany, sebastian.graf@kit.edu.

2022. 2475-1421/2022/1-ART1 $15.00
https://doi.org/